Regular Article

Exploiting Context-Aware Event Data for Fault Analysis

Cuong Huy Nguyen, Ha Manh Tran, Quy Tran Vu, Synh Viet Uyen Ha

School of Computer Science and Engineering, International University, Ho Chi Minh City, Vietnam

Correspondence: Tran Manh Ha, tmha@hcmiu.edu.vn

Communication: received 22 December 2015, revised 10 July 2016, accepted 26 July 2016
Online publication: 10 October 2016, Digital Object Identifier: 10.21553/rev-jec.139
The associate editor coordinating the review of this article and recommending it for publication was Prof. Nguyen Linh-Trung.

***Abstract*– Fault analysis in communication networks and distributed systems is a difficult process that heavily depends on system administrator's experience and supporting tools. This process usually requires analytic techniques and several types of event data including log events, debug messages, trace obtained from these systems to investigate the root cause of faults. This paper introduces an approach of exploiting context-aware data and classification technique for improving this process. This approach uses both event data and context-aware data including CPU load, memory, processes, temperature, status to train a decision tree, and then applies the tree to assess suspected events. We have implemented and experimented the approach on the OpenStack cloud computing system with the Hadoop computing service and MELA event collection system. The experimental results reveal that the accuracy score of the approach reaches 85% on average. The paper also includes detailed analysis for the results.**

***Keywords*– Context-Aware Data, Decision Tree, Fault Analysis, Fault Detection, Cloud Computing.**

1 Introduction

The rapid development of network and storage technology results in the formation of large and complex communication networks and distributed systems that provision computing and storage service solutions for enterprises and individuals, such as data centers, cloud computing systems, content delivery networks, software defined networks, etc. These systems share the common characteristic of associating a large number of network devices and servers with the diversity of configuration and service. The study of Armbrust *et al.* [1] has emphasized 10 obstacles for building and managing large and complex cloud computing systems. Several obstacles belong to fault management including such as fault detection and resolution in large-scale distributed systems, services monitoring, performance monitoring.

Detecting faults occurring on large and complex systems heavily depends on supporting tools and systems administrators. Supporting tools for event monitoring and correlation filters and reports a large number of events to system administrators who then perform investigation steps to detect faults and produce fault reports. Due to the increasing size of event datasets that can easily reach gigabytes per day, the former process faces a problem of efficiency, while the later process is a human driven process that consumes much time and effort. There is a demand of developing an approach that can exploit events obtained from these systems and detect suspected faults for system administrators.

Among several studies of event analysis, the study of Tran *et al.* [2] has applied the classification and regression decision trees (CART) [3] for evaluating the severity level of events automatically, thus enabling system administrators to decide whether further steps are needed for detecting faults. The approach focuses on constructing a CART decision tree based on the features of existing events and then using the tree to determine the severity level of occurring events. However, this study uses software bug datasets and bug features obtained from bug tracking systems (BTSs) to build decision trees. The bug dataset is less related to log events due to the lack of event features that can cause the inaccuracy of event classification. We have proposed an approach of exploiting context-aware data associated with event log data for fault detection. The context-aware data also considered as the metric data specifies the state of a system, such as processor state, memory state, storage state, process state, etc. This approach also enriches runtime monitoring data by capturing events at multiple levels when fault conditions are fulfilled. The approach then uses this dataset as input to construct a decision tree for event classification. The supplement of context-aware data and multi-level event data can improve the accuracy of classifying the severity level of occurring events. The contribution is thus threefold:

- Proposing an approach of using context-aware data for fault detection on cloud computing systems;
- Applying classification technique to improving the accuracy of event classification;
- Providing the performance evaluation of the approach on the OpenStack cloud computing platform with the Hadoop computing service and MELA event collection system [4].

The rest of the paper is structured as follows. Section 2 includes an overview of context-aware data, and presents some existing approaches of context-aware data analysis for fault detection. Section 3 introduces a fault analysis system that can be applied to detect faults on cloud computing systems. This section describes the detail of system architecture and components. Section 4 proposes an approach of exploiting log event data and context-aware data for fault analysis. This section describes a method to enrich monitoring data from the existing tools and exploit event features. It also includes a prototyping implementation of the approach. Several experiments in Section 5 report the performance of the approach on various datasets obtained from the real system before the paper is concluded in Section 6.

2 Background

Many research studies have focused on context-aware computing in the literature. Schilit and Theimer [5] have first introduced the context-aware term to refer to context as location, identity of nearby people and objects, and changes to those objects. This definition of the context is difficult to apply to an application scenario. The study [6] defines context as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application. This definition makes it easier for an application developer to enumerate the context for a given application scenario. The study also defines a system as context-aware if it uses context to provide relevant information and services to the user, where relevancy depends on the user's task. Context awareness has been applied to many area in the social awareness, in health care, in entertainment and in computer application. Hong *et al.* [7] has summarized research activities in context-aware systems and provided suggestions for system implementations. The study of these authors [8] has proposed an agent based approach for predicting user preferences and providing personalized services. This approach has exploited the relationships between user's profile and services on context-aware computing.

Several research studies have applied context-aware computing to detecting faults on large communication networks and distributed systems. Benerecetti *et al.* [9] has presented four steps of processing context information in distributed systems: capture, inference, distribution and consumption. These steps can be applied to analyze actual applications. The study [10] proposes a nine-state model of adaptive behaviour to enable fault detection in applications running on mobile devices. This model detects faults caused by erroneous adaptation logic, and asynchronous update of context information, which leads to inconsistencies between the external physical context and internal representation within an application. The study [11] proposes a dynamic adaptation model that offers increased expressive power to compose complex adaptation rules, and guarantees soundness in fault detection. In addition, the model includes an incremental rule evaluation technique to cater for context-aware applications, such that it can efficiently handle environmental changes in fault detection.

The study [4] introduces an elasticity analytic technique for cloud services. It also defines the concepts of elasticity space and elasticity pathway, and applies these concepts in evaluating the elasticity of cloud services. First, the elasticity space is used for capturing the elastic behaviour of cloud services. Second, the elasticity pathway characterizes the service's evolution through the elasticity space, and can be used for predicting the service's behaviour. This study presents a mechanism for constructing multi-level service monitoring snapshots, over which it applies techniques for determining the elasticity space and pathway. The MELA tool as the result of this study is an open source tool for monitoring and analysing the elasticity of cloud services. It contains a core MELA service, and data collector nodes. A data collector node is a customisable component that gathers, from existing monitoring solutions, e.g., Ganglia, event data associated with a dependency model level or monitored element, e.g., response time or throughput for the event processing service topology, and sends it to the MELA service for processing and analysis. An important MELA feature is the linking of all levels, defining the service structure, applying the metric composition rules to monitored elements at each level in order to provide composite monitoring snapshots from data collected from the data collector nodes for analysing elasticity space.

3 Fault Detection System

Network monitoring systems usually generate a large number of log events that can be difficult to be processed manually, thus causing the difficulty of detecting faults. Using supporting tools to filter trivial events, system administrators also spend much time and effort investigating the correctness and severity of suspected events that can result in fault reports. A fault detection system aims at assisting system administrators in correlating events efficiently using filtering techniques and evaluating the severity level of suspected events using decision trees. This system can be applied for detecting faults on cloud computing systems.

3.1 System Architecture

The system architecture shown in Figure 1 contains a P2P network characterized by self-organization and scalability in architecture and efficiency in content distribution. This network of peers also facilitates the search and sharing of data because queries can be processed by groups of peers on large domain-specific databases, thus avoiding high computing cost on centralized servers. Each cloud system contains a fault manager working as a peer for monitoring. Fault managers connect together through the Gnutella protocol [12] to form a fault detection system. A fault

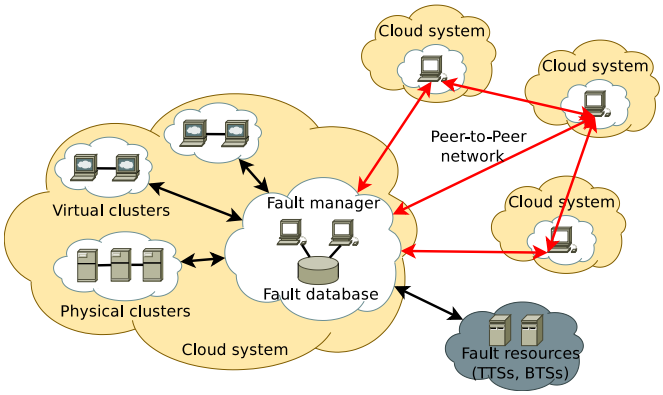

Figure 1. The architecture of the fault detection system [2].

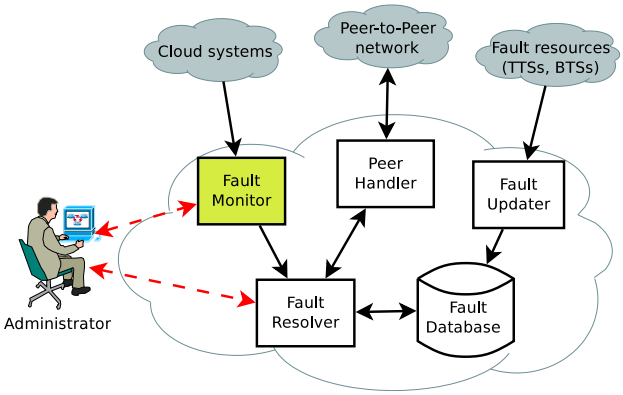

Figure 2. The component communication of the fault manager.

manager needs sufficient storage, bandwidth and processing power to perform several complicated operations, such as search and share fault resources, receive update on fault datasets, monitor and evaluate suspected events. Our previous study [2] has proposed the fault detection system in communication networks and distributed systems. While the previous study uses software bug reports to evaluate suspected events, this study aims at exploiting both context-aware data and event to evaluate suspected events.

The fault manager contains several components shown in Figure 2: fault monitor, fault checker, fault updater, fault database and peer handler. The fault monitor uses monitoring tools such as Nagios [13], Cacti [14], Ntop [15], Ganglia [16] to obtain a huge number of log events from cloud systems. Log events are related to several functional areas of services, networks, servers, clusters. This component also cooperates with the MELA system to obtain context-aware event data. It contains an efficient event correlation mechanism to filter correlated events. The fault checker uses a classification method to assess the severity of the filtered events sent by the fault monitor. This method obtains fault datasets from the local fault database and other fault managers through the P2P network to construct a decision tree. This component reports a list of suspected events to system administrators for additional actions. The peer handler is responsible for communicating among peers. This component searches and shares fault reports among peers by exchanging query and queryhit messages. The fault updater contains fault report crawlers that connect to multiple fault resources such as bug tracking systems, archives, forums, vendor knowledge bases to update the fault database.

3.2 Event Correlation and Inspection

We have applied the ASF-BDT method for event correlation [17] and the CART decision tree for event inspection [2]. The correlation method contains multiple filtering processes: starting with log event data, applying with simple and complex filtering methods, finally returning event correlated data. The log event data usually contains a large number of related and duplicated events. The method thus eliminates duplicated and related events efficiently with low time consumption. The inspection method contains a binary recursive partitioning process to build a decision tree. This process starts with the root node where data are split into two children nodes and each of the children node is in turn split into grandchildren nodes. The process runs recursively until no further splits are possible due to lack of data and the tree reaches a maximal size. This tree is then used for evaluating the severity level of suspected events.

4 Context-Aware Data Analysis

As the fault detection system is described above, we propose a data analysis framework for cloud systems. This proposal monitors and collects event data from cloud services running on the Hadoop cluster and the OpenStack platform. Structuring and analysing the monitoring data are then performed on the MELA server before event correlation and classification. Figure 3 plots the detailed architecture of the OpenStack cloud system monitored and analysed by the MELA system. The cloud system is built based on RedHat OpenStack platform. It includes 8 nodes that connect together through the local area network. The controller node provides multiple services including identity service (*keystone*), computing service (*nova*), web interaction server (*horizon*), image service (*glance*) and data processing service (*sahara*). The network node provides centralized networking control (*neutron*) for all the computing nodes and virtual machines. It runs various neutron agents that control layer-3 networking functionality in the cluster. The storage nodes provide storage service (*swift*) that can store, retrieve, and delete objects stored on local devices. These objects are stored using a path derived from the object's name and timestamp. The computing nodes refer to an OpenStack server that runs a KVM hypervisor. It is responsible for running virtual machine instances.

Sahara is the OpenStack project that provides a scalable data processing service and associated management technologies for provisioning Hadoop clusters. We deploy the Hadoop cluster by specifying several parameters like Hadoop version with vanilla plug-in version 2.4.1, cluster topology with 1 master node and 3

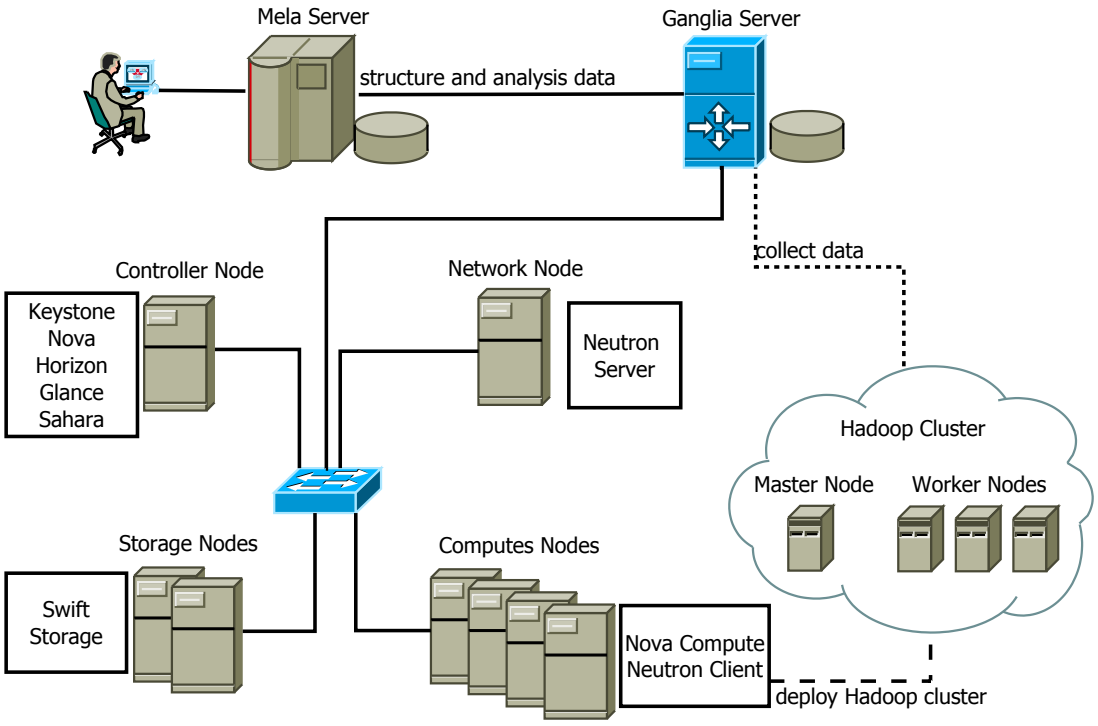

Figure 3. Context-aware data analysis framework for cloud systems.

worker nodes. These nodes are deployed as instants running on 4 compute nodes. After filling all the parameters, Sahara automatically operates on the cluster. MELA is a tool for structuring and enriching event log data collected from existing monitoring solutions. We need monitoring services to collect the data from all the nodes in the Hadoop cluster as the source of MELA. In this study, we also use Ganglia servers to collect metric data of the Hadoop cluster. The metric data or context-aware data includes 6 metrics: CPU usage, disk usage, load average, memory usage, network throughput and running processes that specify the state of the cluster nodes.

4.1 Data Collection

MELA data contains monitoring snapshots that are the result of applying cross-layer metric composition rules to monitoring data collected from Ganglia servers and log events data for specific elements. As shown in Figure 3, Ganglia servers monitor and collect 6-metric data from all the nodes in the Hadoop cluster including 1 master node and 3 worker nodes, while log events are collected by the MELA server and structured into cloud service model. Some composition rules are applied to this model to create monitored snapshots, such as calculate average CPU usage or summarize running processes of all the worker nodes.

MELA data is exported from the MELA server in XML format file with a complex structure. It also contains several trivial fields such as freshness, hashCode. We use XMLStarlet [18], a set of command line XML utilities, to browse, query, transform, validate, search and edit the tree structure of XML documents, and then extract the 6-metric data from the raw data. A MELA log report contains 6 features corresponding to 6 metrics. Since it only contains metrics, it misses the severity feature. We define the severity feature based on the average CPU usage with *error*, *warn* and *info* level, e.g., an *error* occurs if the average CPU usage is between 85% and 100% and a *warn* occurs if the average CPU usage is between 70% and 84%.

We have used a MELA dataset with 100.000 log reports for experiments, the dataset is collected with an interval of 5 seconds to capture changes on the system, but it also produces many duplications on idle time. It is essential to reduce duplicated reports in the MELA dataset before constructing a decision tree. We have used the *uniq* build-in Linux utility to remove duplicated reports for convenience and efficiency. It only takes a few minutes to reduce the dataset to 27.000 log reports.

4.2 Cloud Service Model

In the complex cloud system, cloud services, e.g., network service or storage service, can run on multiple distributed virtual machines (VMs). Depending on the service requirement of storage space and performance improvement during run-time, the cloud controller can re-scale the service by adding or removing VMs to fulfil the requirement. The existing monitoring tools, such as Ganglia, focus on monitoring data for an individual virtual machine that can be modified during service execution, it is thus inefficient to support scalability for the controller nodes. Moreover, the users of the cloud services can be interested in various monitoring data at multiple levels, such as average CPU usage or total network bandwidth on specific services.

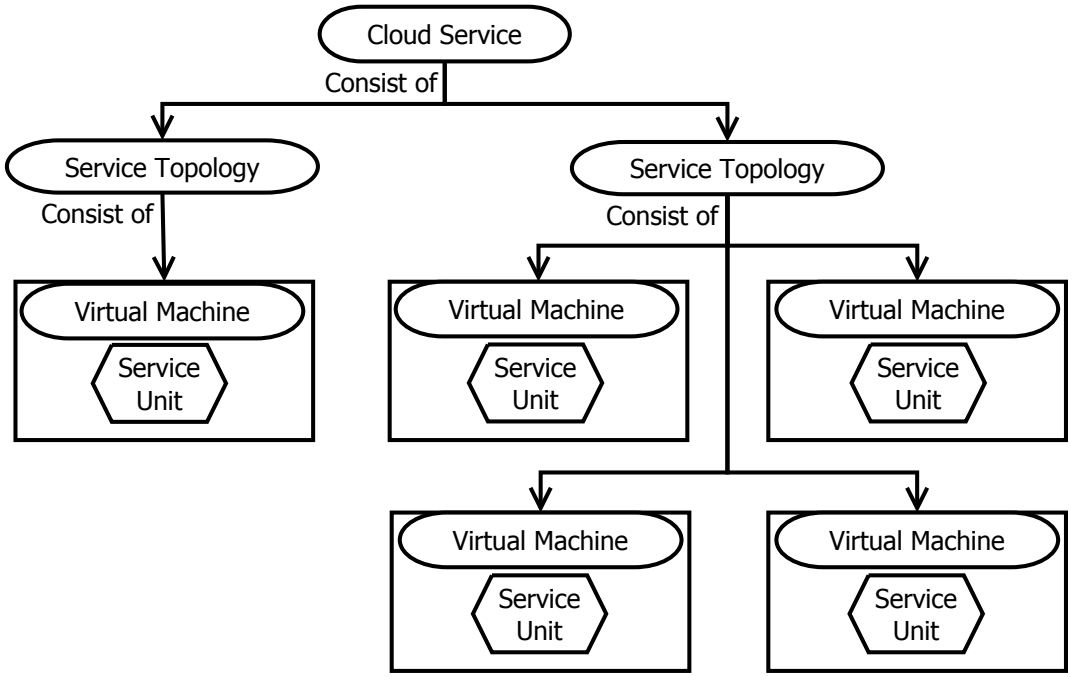

Figure 4. Cloud service model.

We need to collect, store and integrate monitoring data into a model in which service unit is the core monitored entity, not only virtual machine. A cloud service model is defined to structure monitoring data and analyse cloud services from the cloud service level to the virtual infrastructure level. This model contains 4 levels as shown in Figure 4: cloud service, service topology, service unit and virtual machine. A cloud service contains service topologies; each service topology includes service units deployed on virtual machines. A service unit is the core monitored element of a cloud service that runs on virtual machines, either standalone or along with other service units. A service topology logically groups related service units.

Listing 1 presents an example of cloud service in XML format. Each monitored element contains id and level properties. The level property receives values of SERVICE, SERVICE_TOPOLOGY, SERVICE_UNIT and VM. If the element level is VM, the id is the IP address of the VM.

Listing 1. An Example of Cloud Service in XML Format

```
<MonitoredElement id="WorkerTopology"
    level="SERVICE_TOPOLOGY">
  <MonitoredElement id="MasterNode"
      level="SERVICE_UNIT">
    <MonitoredElement id="10.0.2.176"
        level="VM"/>
  </MonitoredElement>
  <MonitoredElement id="WorkerNode"
      level="SERVICE_UNIT">
    <MonitoredElement id="10.0.2.177"
        level="VM"/>
    <MonitoredElement id="10.0.2.178"
        level="VM"/>
    <MonitoredElement id="10.0.2.179"
        level="VM"/>
    </MonitoredElement>
  </MonitoredElement>
</MonitoredElement>
```

4.3 Cross-Layered Metric Composition Rule

Monitoring data is captured from each monitored element to create monitoring snapshots at specific points of time. Monitoring snapshot only captures metrics such as responseTime, CPUUsage, throughput from individual VMs. It does not provide the performance state of whole cloud service or the fulfilled fault requirement. Cross-layered metric composition technique creates new multiple level metrics by associating metrics from VM level to upper service level. Listing 2 presents an example of metric composition rule in XML format.

Listing 2. An Example of Metric Composition Rule in XML Format

```
<ResultingMetric type="RESOURCE"
    measurementUnit="%" name="cpuUsage
    "/>
<Operation value="100" type="ADD">
  <Operation value="-1" type="MUL">
    <Operation MonitoredElementLevel
        ="VM" type="AVG">
      <ReferenceMetric type="
          RESOURCE" measurementUnit
          ="%" name="cpu_idle"/>
    </Operation>
  </Operation>
</Operation>
</CompositionRule>
```

A composition rule includes a target monitored element that applies the rule, a resulting metric

Table I
Metric Composition Operations

Operation	Description
ADD	Add a metric value to other metric value
DEL	Subtract a metric value from other metric value
MUL	Multiple a metric value with other metric value
DIV	Divide a metric value from other metric value
AVG	Compute the average value of many metric values
SUM	Sum many metric values
MAX	Compute the maximum value from many metric values
MIN	Compute the minimum value from many metric values
SET	Assign a value to a metric value
KEEP	Return a metric value or the result of another operation

that is the output of the rule, and several operations. First, *CompositionRule* defines the level (SERVICE, SERVICE_TOPOLOGY, SERVICE_UNIT, VIRTUAL_CLUSTER, and VM) in *TargetMonitoredElementLevel* that the rule is applied. Second, *ResultingMetric* defines the resulting data of the rule including type, measurement unit and name. Finally, the list of operations is applied at the specified *MonitoredElementLevel*. Note that the result of the lower operation is the input of the higher operation in the composition rule. Table I shows operations can be used in the composition rule.

4.4 Monitoring Data Structure and Enrichment

User requirement for monitoring services can be different from monitoring snapshot that captures metrics, e.g., a monitoring snapshot includes throughput over VM, while requirement targets performance over the whole service. Using the above metric composition rule, metrics collected from the VM level are associated with the upper service levels, and therefore new metrics are created. First, the level of the new metric is specified by *TargetMonitoredElementLevel* that is a value of VM, VIRTUAL_CLUSTER, SERVICE_UNIT, SERVICE_TOPOLOGY, or SERVICE according to the cloud service model. The id of the target monitored element is specified by *TargetMonitoredElementID*. Data of the new metric must be specified using *ResultingMetric* including name, measurement unit, and type (such as performance or resource). Then, a cascading list of operations is defined; each operation contains a type defined in Table I, and *ReferenceMetric* indicating the metric over which the operation is applied. If an operation applies to metrics from a specific monitored element, the id is specified with *SourceMonitoredElementID*.

5 Evaluation

We have configured the OpenStack platform [19] to provide an infrastructure as a service for experiments. The platform contains 10 nodes equipped with Intel Core i5 2.8 GHz processor, 4 GB RAM and Redhat Linux operating system. The cloud controller node provides several OpenStack services including computing, dashboard, object storage, image and data processing. The network node provides centralized networking

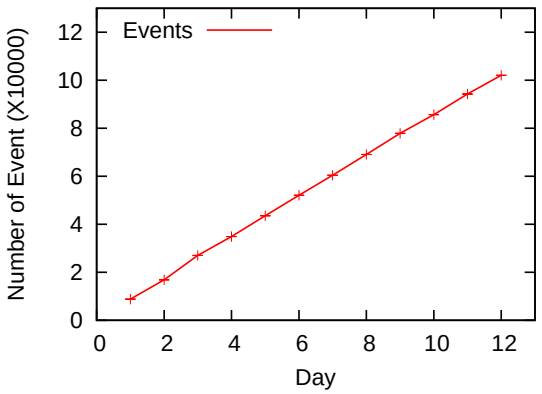

Figure 5. Data collection for a period of 12 days.

control for all the computing nodes running a KVM hypervisor [20]. The Hadoop [21] cluster instantiated across multiple computing nodes accepts several computing tasks and generates log datasets. Ganglia [16] obtains monitoring data from Hadoop nodes while MELA arranges the data on cloud service model and also composes monitored snapshots by composition rules.

We have configured Ganglia to collect 6 types of Ganglia metrics for evaluation: CPU usage, disk usage, load average, memory usage, network throughput and running processes. These metrics are collected from all the worker nodes and then associated with log events to present monitored snapshots by specific composition rules, such as measuring average CPU and memory usage, or checking running processes on worker nodes. Since the dataset only contains metrics without the severity feature, we have pre-defined the severity feature based on the average CPU usage of the whole system with *error*, *warn* and *info* levels. We have created several failure scenarios while executing computing tasks to obtain log data. Some typical scenarios include submitting several jobs in parallel to increase the workload of the cluster or manually interrupting few worker nodes to cause the failure of the job assignment. Figure 5 reports the dataset collected by a period of 12 days. The data records stably increase everyday and reach 100.000 records on the last day. We use this log dataset for evaluating the performance of CART tree.

The first experiment measures time consumption for constructing decision trees over various datasets. Figure 6 shows that time consumption linearly increases as the size of the dataset increases. It takes approximately 280 ms to build a decision tree for the dataset of 100.000 records. Note that the log dataset usually contains millions of log records, causing a large tree with high time consumption. Reducing processing time is necessary. In addition, this dataset possibly contains duplicated records that need to be eliminated to improve performance. By filtering out duplicated records, the dataset contains 27.000 records that takes 60 ms to build a decision tree approximately.

The second experiment evaluates the accuracy of the decision tree. The dataset is divided into two training

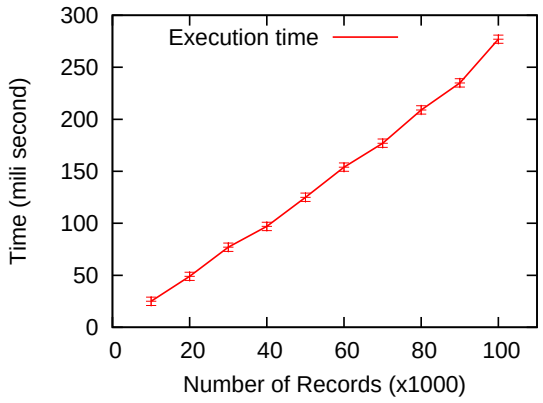

Figure 6. Time consumption for constructing decision trees over various datasets.

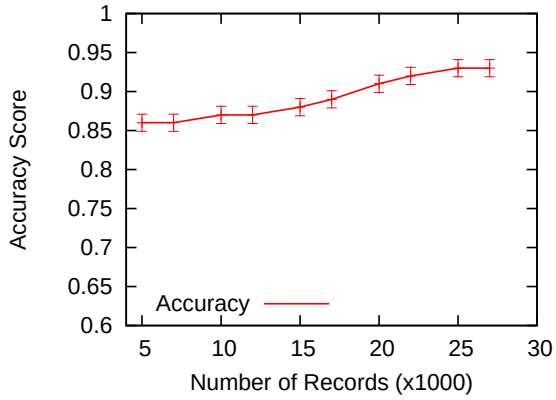

Figure 7. The accuracy score of the decision tree.

and testing datasets. The training dataset is used to build the decision tree, and the testing dataset is used to assess the accuracy of classification. We have used the decision tree to classify the severity level of a record and make a list of the classified severity levels for the testing dataset (MELA records). This list is then compared with the list of the correct severity levels of the testing dataset. Accuracy score is calculated based on the number of matching severity levels in the two lists.

Figure 7 reports the accuracy score of the decision tree. Accuracy score starts with 0.86 at 5.000 records, slowly increases when the size of the dataset increases, and reaches 0.93 at 27.000 records. An observation reveals that providing more log records collected from the real system can make the decision tree larger and more accurate. The previous study [2] has also built the decision tree based on log events and obtained the maximum accuracy score of 0.71. Our method considers both log events and context-aware log records for improving accuracy.

The third experiment evaluates the quality of decision tree using precision and recall. Precision presents the exactness of a classifier, while recall presents the completeness of a classifier. A decision tree with high recall but low precision returns many results of a specific class, but most of the results are incorrect when

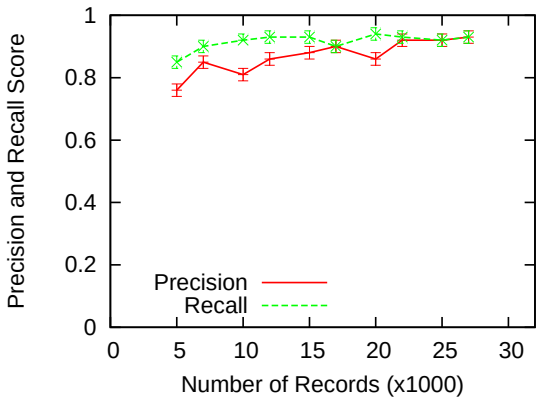

Figure 8. The precision and recall score of the decision tree.

compared with testing dataset. Otherwise, with high precision but low recall, the decision tree returns few results, but most of the results are correct. A decision tree with high precision and high recall returns many results classified correctly.

Figure 8 shows the precision and recall score of the decision tree. The precision and recall scores are considerably high for the dataset. While the precision score fluctuates slightly, the recall score remains stably as the size of the dataset increases. The average precision and recall scores are 0.9 and 0.85, respectively. However, it is essential to evaluate the approach on large datasets.

6 Conclusion

We have proposed an approach of using context-aware data and classification technique for evaluating the severity level of the suspected log events. This approach aims at assisting system administrators in detecting faults from a large number of log events generated by large and complex communication networks and distributed systems.

We have implemented and configured the approach on the OpenStack cloud computing platform with the Hadoop computing service and MELA system. The approach applies monitoring data to the cloud service model, then uses cross-layer composition rules to extract and enrich monitoring snapshots at multiple levels when fault conditions are fulfilled. This data is considered as input to construct the CART decision tree for evaluation. The system administrators then take additional actions on a certain set of the suspected events that may result in serious failures. We have evaluated the performance of the proposed approach and CART decision tree on both log event data and context-aware data collected from the cloud system. The experimental results show that this approach has achieved the high accuracy score of 85% compared to the previous approach [2] with the accuracy score of 71% on average.

Future work focuses on extending the approach to large datasets with various context-aware data.

Acknowledgements

This research activity is funded by Vietnam National University, Ho Chi Minh City (VNU-HCM) under the type-B project of "Augmenting fault detection services on large and complex network systems using context-aware data analysis" in 2017.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] H. M. Tran, S. Van Nguyen, S. T. Le, and Q. T. Vu, "Fault Data Analytics Using Decision Tree For Fault Detection," in *Proceedings of the 2nd International Conference on Future Data and Security Engineering (FDSE 2015)*. Springer-Verlag, 2015.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification And Regression Trees*. Monterey, CA, USA: Chapman & Hall/CRC, New York, 1984.

[4] D. Moldovan, G. Copil, H. L. Truong, and S. Dustdar, "MELA: Monitoring and Analyzing Elasticity of Cloud Services," in *Proceedings of the 5th International Conference on Cloud Computing*. IEEE Press, 2013, pp. 80–87.

[5] B. Schilit and M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *IEEE Network*, vol. 8, no. 5, pp. 22–32, 1994.

[6] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," in *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC'99)*. London, UK: Springer-Verlag, 1999, pp. 304–307.

[7] J. Hong, E. Suh, and S. Kim, "Context-aware systems: A literature review and classification," *Expert Systems with Applications*, vol. 36, no. 4, pp. 8509–8522, May 2009.

[8] J. Hong, E. Suh, J. Kim, and S. Kim, "Context-aware system for proactive personalized service based on context history," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7448–7457, May 2009.

[9] M. Benerecetti, P. Bouquet, and M. Bonifacio, "Distributed context-aware systems," *Human-Computer Interaction*, vol. 16, no. 2, pp. 213–228, Dec. 2001.

[10] M. Sama, D. Rosenblum, Z. Wang, and S. Elbaum, "Model-based Fault Detection in Context-aware Adaptive Applications," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, USA: ACM, 2008, pp. 261–271.

[11] C. Xu, S. Cheung, X. Ma, C. Cao, and J. Lu, "Detecting Faults in Context-Aware Adaptation," *International Journal of Software and Informatics*, vol. 7, no. 1, pp. 85–111, 2013.

[12] "Gnutella Protocol Specification version 0.4," http://rfc-gnutella.source-forge.net/developer/stable/index.html, 2001, last access in Nov. 2015.

[13] "Nagios Network Surveillance Tool," http://www. nagios.org/, 1999, last access in Nov. 2015.

[14] "Cacti - The complete network graphing solution," http://www.cacti.net/, 2004, last access in Nov. 2015.

[15] "High performance network monitoring solution," http://www.ntop.org/, 1998, last access in Nov. 2015.

[16] "Ganglia Monitoring System," http://ganglia.info/, 2000, last access in Nov. 2015.

[17] H. M. Tran, A. V. T. Tran, S. T. Le, and S. V. Nguyen, "Improving Adaptive Semantic Filtering with Bounded Dynamic Threshold for Log Data Analytics," *Journal of Science and Technology, Vietnamese Academy of Science and Technology*, 2014.

[18] "XMLStarlet Command Line XML Toolkit," http:// xmlstar.sourceforge.net/, 2016, last access in Mar. 2016.

[19] "OpenStack Cloud Software," http://www.openstack.org/, 2010, last access in Nov. 2015.

[20] "Kernel based virtual machine," http://www.linux-kvm.org, last access in Nov. 2015.

[21] "Apache Hadoop Project," http://hadoop.apache.org/, 2005, last access in Nov. 2015.

Cuong Huy Nguyen is an IT engineer and nework administrator of Renesas Design Vietnam Company. He received his bachelor degree of information technology in 2004 and his master degree of information technology in 2016 from the International University - Vietnam National University. His research interests include distributed computing, big data analytics and network management.

Ha Manh Tran is a lecturer of the School of Computer Science and Engineering at International University - Vietnam National University. He received his master degree of computer science in 2004 from the University of Birmingham, United Kingdom and his doctoral degree of computer science in 2009 from Jacobs University Bremen, Germany. His research interests include distributed computing, big data analytics, information retrieval and network management.

Quy Tran Vu is a research assistant of School of Computer Science and Engineering at International University - Vietnam National University. He received his bachelor degree of computer science in 2016 from the International University - Vietnam National University. His research interests include machine learning, distributed system and network management.

Synh Viet Uyen Ha is a lecturer of the School of Computer Science and Engineering at International University - Vietnam National University. He received his master degree of computer science in 1999 from University of Natural Science, Vietnam and his doctoral degree of computer science in 2010 from School of Information and Communication Engineering, Sungkyunkwan University, Korea. His research interests include machine learning, computer vision, big data processing.